OULUN SEUDUN
AMMATTIKORKEAKOULU

Joonas Hannula

WINDOWS-TEKNIIKOIDEN SOVELTAMINEN USEAN ALUSTAN TUOTEKEHITYKSESSÄ

WINDOWS-TEKNIIKOIDEN SOVELTAMINEN USEAN ALUSTAN TUOTEKEHITYKSESSÄ

Joonas Hannula
Opinnäytetyö
Kevät 2013
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä: Joonas Hannula
Opinnäytetyön nimi: Windows-tekniikoiden soveltaminen usean alustan tuotekehityksessä
Työn ohjaaja: Tuomo Tikkanen
Työn valmistumislukukausi ja -vuosi: Kevät 2013 Sivumäärä: 37

Työn tavoitteena oli tutustua erilaisiin Windows-tekniikoihin ja laatia niiden pohjalta soveltamisohjeet, joiden avulla tekniikoita voidaan hyödyntää mobiilin toiminnanohjausjärjestelmän tuotekehityksen erityisvaatimusten toteuttamisessa. Työn tilaajana toimi oululainen mobiilikehittäjä M-Technology Oy.

Soveltamisohjeita kokeiltiin käytännössä WiseMaster Huolto- ja WiseMaster Myynti -tuotteiden tuotekehityksessä. Huollosta toteutettiin Windows RT -versio, Myynnistä puolestaan Windows Phone 7-, Windows 8- ja Windows RT -versiot sekä kokonaan uusi serveri.

Tärkein Windows-tekniikka, jota projekteissa sovellettiin, oli Model-View-ViewModel (MVVM) -kehitysmalli. Tätä kokeiltiin yhdessä Portable Class Libraries (PCL) -luokkakirjaston kanssa. MVVM-mallin käyttö todettiin hyödylliseksi, jos projektia kehitti yhtä aikaa monia henkilöitä usealle eri alustalle. Pienemmissä projekteissa malli ei kuitenkaan tuonut merkittävää lisäetua.

Testauksen merkitys korostui usean alustan tuotekehityksessä. Projekteissa ei sovellettu Test-driven development (TDD) -testausmallia, mutta sen käyttö tulevissa tämän kaltaisissa projekteissa voi osoittautua hyödylliseksi.

Windows 8 antaa käyttäjälle mahdollisuuden käyttää näyttöä eri tiloissa (vaakasuunnassa, pystysuunnassa ja 320 pikselin levyisenä), ja tämä tulee ottaa huomioon tuotekehityksessä. Käyttöliittymiin on muutenkin kiinnitettävä erityistä huomiota, jotta käyttökokemus eri alustojen välillä olisi mahdollisimman saumaton.

Soveltamisohjeiden todettiin olevan varsin päteviä, ja niitä kannattaakin hyödyntää myös tulevissa projekteissa.

Asiasanat: ohjelmistokehitys, Windows 8, Windows Phone, taulutietokoneet

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software Development

Author: Joonas Hannula
Title of thesis: Utilization of Windows Techniques in Multi-Platform Product Development
Supervisor: Tuomo Tikkanen
Term and year of completion: Spring 2013 Pages: 37

The objective of this thesis was to explore different Windows techniques and create application directives for making the most of them in the product development of mobile enterprise resource planning system. The orderer of this thesis was mobile developer M-Technology Oy from Oulu.

The application directives were applied in the development of the products WiseMaster Huolto (Service) and WiseMaster Myynti (Sales). A Windows RT version was developed from WiseMaster Huolto, and Windows Phone 7, Windows 8 and Windows RT versions were developed from WiseMaster Myynti. A completely new server was also developed for WiseMaster Myynti.

The most important Windows tehcnique that was applied in these projects was Model-View-ViewModel (MVVM). This was tested together with Portable Class Libraries (PCL). MVVM was proved useful, if project had multiple designers and multiple target platforms. However, in smaller projects MVVM did not bring enough extra benefit to be truly useful.

Testing was another thing that showed to be truly important. These projects did not use Test-driven development (TDD), but it could be useful in future projects of these kind.

Thirdly user interface needed extra attention in multi-platform product development. Windows 8 allows user to use different kind of views (landscape, portrait and snapped view) and this needs to be taken into account when program is designed. Also user experience must be as seamless as possible when switching between different platforms.

The application directives proved to be quite useful and they should be used also in future projects.

Keywords: software development, Windows 8, Windows Phone, tablet

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tavoitteena oli tutustua erilaisten Windows-tekniikoiden soveltuvuuteen mobiilin toiminnanohjausjärjestelmän tuotekehityksessä. Kohdealustoina toimivat Windows 8 -käyttöjärjestelmällä varustettu pöytäkone, Windows Phone 7 -puhelin sekä Windows RT -tabletti. Työn tilaajana toimi oululainen mobiilikehittäjä M-Technology Oy.

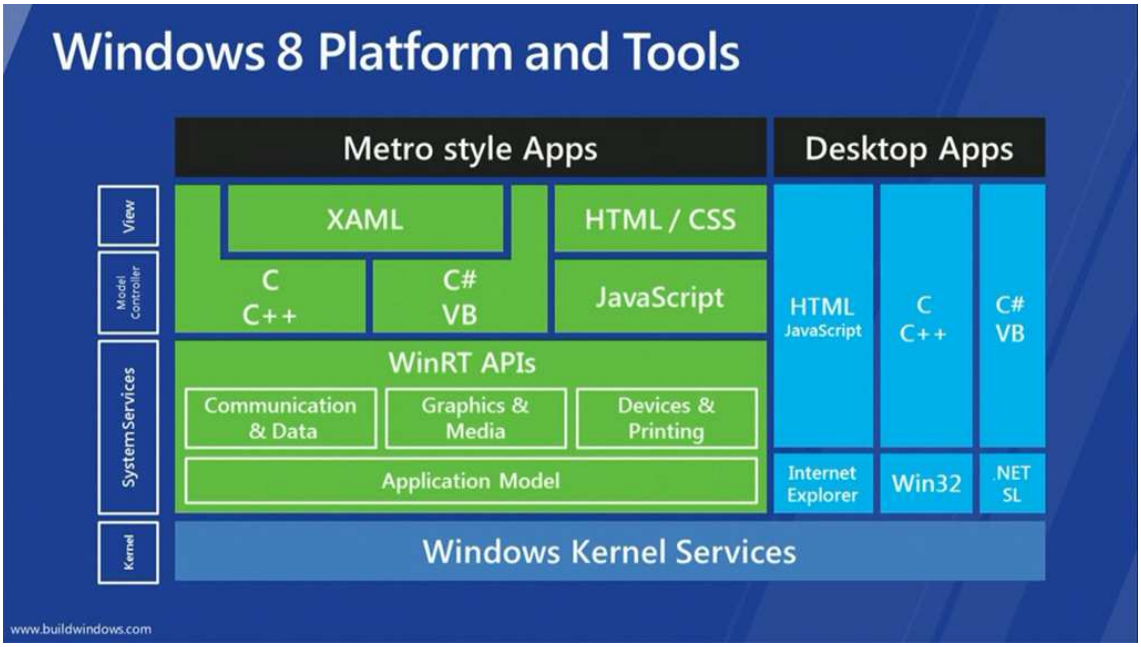
Työssä tutkittiin mobiilin toiminnanohjausjärjestelmän erityisvaatimuksia ja laadittiin ohjeet siitä, miten eri Windows-tekniikoita voidaan soveltaa kyseisten vaatimusten toteuttamisessa. Lopuksi suoritettiin analyysi projektin kokonaistuotoksesta ja pohdittiin, miltä osin eri tekniikoita voidaan hyödyntää tämän kaltaisissa projekteissa.

Opinnäytetyössä muodostettujen soveltamisohjeiden käyttöarvoa analysoitiin M-Technologyn WiseMaster Myynti- ja WiseMaster Huolto -tuotteiden kehitysprosesseissa. Ohjelmista on olemassa aikaisemmat versiot, mutta projektin aikana ohjelmat toteutettiin kokonaan uudestaan.

2 WINDOWS-OHJELMOINNIN TEKNIIKAT

2.1 Arkkitehtuuri

Windows 8 tarjoaa kehittäjille kaksi erilaista ohjelmatyyppiä: Metro-tyyliset ohjelmat ja työpöytäohjelmat (kuva 1). Työpöytäohjelmat toimivat PC-alustoilla kuten ohjelmat tähänkin asti. (1.) Metro puolestaan on Microsoftin uusi tyylisuunta, jonka avulla ohjelmat ovat samankaltaisia eri alustoilla (2). Microsoft on Windows 8:n julkaisemisen jälkeen luopunut sanan Metro käytöstä, ja siitä käytetään yleisesti vain nimitystä "Modern UI" (3) ja sen ohjelmista nimitystä ”Windows Store App” (4).



KUVA 1. Windows 8:n arkkitehtuuri (1)

Modern UI:n periaatteet voidaan tiivistää viiteen perusperiaatteeseen. Ensinnäkin ohjelmien tulisi olla autenttisesti digitaalisia eli niiden tulisi keskittyä toiminnallisuuteen sen sijaan, että ne mukailisivat oikean maailman vastaavuuksia. Esimerkiksi e-kirjan sivun käännön tulisi siten olla nopea ja helppokäyttöinen eikä käyttää oikean kirjan kaltaista sivunkääntöanimaatiota. Toisena periaatteena ohjelman tulisi olla hyvännäköinen. Yksityiskohtiin tulee kiinnittää huomiota eikä ohjelman ulkoasussa tulisi olla moittimista. Kolmantena ohjelman tulee olla nopea ja sujuva. Ohjelman on vastattava komentoihin tarkasti. Neljänneksi oh-

jelman tulee olla mahdollisimman yksinkertainen. Esimerkiksi ruudulla näytettävien nappien määrän tulee olla mahdollisimman pieni, jottei näkymä alkaisi vaikuttaa liian ahtaalta. Viidenneksi ohjelman tulee käyttää Microsoftin kehittämiä designohjeita, jotta kynnys ohjelman käyttämiseen olisi mahdollisimman matalalla. (2.)

2.2 Model-View-ViewModel

Model-View-ViewModel (MVVM) on Silverlight- ja Windows Presentation Foundation (WPF) -ympäristöille tehty suosittu kehitysmalli, jolla on lukuisia etuja. Mallin ansiosta ohjelmoija ja käyttöliittymän suunnittelija voivat tehdä työtään samanaikaisesti, yksikkötestaus helpottuu huomattavasti, komponentteja voidaan käyttää sellaisenaan eri projekteissa ja käyttöliittymää voidaan parantaa ilman, että toimintakoodiin tarvitsee tehdä muutoksia. (5.)

2.2.1 Toimintaperiaate

Model-View-ViewModel koostuu nimensä mukaisesti kolmesta osasta: modelista, view'stä ja view modelista. Model tarkoittaa käsiteltävää dataa, kuten esimerkiksi puhelimessa olevaa yhteyshenkilöä. Model ainoastaan sisältää datan, mutta ei käsittele sitä millään tavalla. Model ei esimerkiksi päivitä sisältöään itse eikä näytä sisältöään tietokoneen näytöllä. (5.)

View tarkoittaa käyttäjälle näytettävää käyttöliittymää, joka esittää modelin sisältämän datan halutussa muodossa. Model voi esimerkiksi sisältää kellonajan UTC-aikana, mutta view'ssä se näytetään käyttäjän paikallisessa ajassa. View huolehtii myös käyttäjän toiminnoista, kuten näppäimenpainalluksista, hiiren liikkumisesta ja kosketusnäytön toiminnallisuudesta. Modelin tapaan view ei päivitä itse sisältöään, vaan tästä huolehtii view model. (5.)

View modelin tarkoituksena on hoitaa view'n ja modelin yhdistäminen ja siten sisältää varsinainen toimintalogiikka. View model esimerkiksi hakee modelin sisältämän päivämäärädatan, muuttaa sen haluttuun muotoon ja lähettää sen view'lle näytettäväksi. View model myös muuttaa modelin sisältämää dataa käyttäjän view'llä tekemien muutosten perusteella ja hoitaa view'stä tulevat tapahtumat (events). (5.)

View ja view model kommunikoivat keskenään datasidoksilla (data-binding), metodikutsuilla (method calls), ominaisuuksilla (properties), tapahtumilla (events) ja viesteillä (messages). View käsittelee itse omat käyttöliittymää koskevat tapahtumat ja lähettää niistä käskyjen (commands) välityksellä tiedon view modelille. Modelit ja view modelin ominaisuudet päivitetään view'stä kaksisuuntaisella datasidoksella. (5.)

Jotta MVVM-mallista olisi hyötyä, ei yhdellä view'llä tule olla kuin yksi view model. Kuitenkin yhdellä view modelilla voi olla useita view'itä. Asennusopas (setup wizard) on hyvä esimerkki tästä: asennus koostuu useista eri ikkunoista (view), mutta koko prosessia voi ohjata yksi ainoa view model. (5.)

MVVM tarvitsee toimiakseen kaksi asiaa: luokan, joka perii joko DependencyObject- tai INotifyPropertyChanged-luokan kunnollisen datasidostuen aikaansaamiseksi, ja jonkinlaisen käskytuen. Silverlight 4 tukee käskyjä sellaisenaan, mutta Silverlight 3 vaatii erikseen ICommand-luokan implementoinnin. (5.)

2.2.2 Model-View-ViewModelin etuja

MVVM-malli on niin tehokas, että Microsoft käytti sitä itse kehittäessään WPF-ohjelmia (kuten kehitystyökalua Microsoft Expression Blend), vaikka itse WPF oli vielä kehitysvaiheessaan. WPF:ssä onkin useita näkökohtia, joissa MVVM-mallin vaikutus näkyy. (6.)

Tärkeimpänä näkökohtana voidaan pitää datasidoksia. Kun ominaisuudet sidotaan view'stä view modeliin, ei view modeliin tarvitse lainkaan kirjoittaa koodia, joka päivittäisi view'n. Datasidokset tukevat myös syötteentarkastusta (input validation), joka tarjoaa standardimenetelmän tarkistusvirheiden lähettämiseen view'lle. (6.)

View model -luokkien yksikkötestaus on helppoa. Koska ohjelman toimintalogiikka on puhtaasti view modeleissa, voidaan view modeleille tehdä täysin view'stä ja modelista riippumattomat testit. View modelille voidaan tehdä testit täysin ilman käyttöliittymää, sillä se ei ole millään tavalla käyttöliittymästä riippuvainen. View modelin nopea regressiotestaus auttaa pitkällä tähtäimellä pienentämään ohjelman ylläpitokustannuksia. (6.)

View’n ja view modelin erottamisen ansiosta käyttöliittymä ja view modelin toimintalogiikka voidaan suunnitella toisistaan riippumatta. Niinpä eri kehittäjät voivat rauhassa hioa ohjelman käytettävyyttä ja toimintavarmuutta. Käytännössä ainoa asia, josta kehittäjien tulee huolehtia view’tä ja view modelia yhdistettäessä, on datasidosten oikea käyttö. (7.)

2.2.3 Yleisiä ongelmanratkaisumenetelmiä

MVVM-malli on perusajatukseltaan melko suoraviivainen, mutta jotkin tietyt erityistilanteet voivat olla ongelmallisia. Tähän on koottu joitakin yleisimpiä ongelmia ratkaisuineen.

Yksi yleisimmistä kysymyksistä MVVM-mallissa on se, miten yhdistelmäruudusta (combo box) voidaan valita yksittäisiä tai useita kohteita (items). Esimerkkinä voidaan käyttää vaikkapa yhteystietoluetteloa, jossa henkilön nimeä painamalla aukeaa näytölle henkilön tarkemmat yhteystiedot. Ratkaisuna yhdistelmäruudulle tehdään kaksi eri datasidosta: ItemsSource ja SelectedItem. Näin ollen koko yhteystietolista sidotaan ItemsSourceen ja valittu henkilö sidotaan SelectedItemiin. Näin toimimalla tarvitaan vain yksi view model, johon yhdistetään kaksi view’iä. (5.)

Toinen tärkeä kysymys on navigoinnin eli sivulta toiselle siirtymisen hoitaminen mahdollisimman tehokkaasti. Navigointiin voi käyttää omaa moottoriaan, joka hoitaa view’ien vaihtamisen, Silverlightin omaa navigointiin tarkoitettua luokkakirjastoa, Prism-tekniikan aluehallintaa tai kaikkien edellä mainittujen tekniikoiden yhdistelmää. Menetelmästä huolimatta mekaniikka tulisi suunnitella käyttöliittymän taakse. INavigation on luokka, jonka avulla view model voi navigoida tai kutsua siirtymistä tarpeen vaatiessa. Navigoinnissa tulee myös ottaa huomioon ohjelman lataaminen. Jos ohjelman koko on suuri, hidastaa kaiken datan yhdellä kertaa lataaminen sen käyttöä merkittävästi. Niinpä ohjelma kannattaa ladata pala kerrallaan, eli data ladataan vasta sitten, kun sitä oikeasti tarvitaan. MVVM ei tarjoa tähän suoraa ratkaisua, mutta ongelmaan voidaan hyödyntää esimerkiksi Prismin tapahtumakokoajaa (event aggrogator). (5.)

Suuret tietokannat voivat väärin käytettyinä olla este MVVM-mallin tehokkaalle käytölle. Ratkaisuna käytetään useimmiten tulosten jakamista usealle sivulle, mutta tämä toteutetaan usein huonosti. Datan jakaminen eri sivuille toteutetaan vain erillisenä jakofunktiona, jolloin kaikki data haetaan kerralla, mutta siitä näytetään käyttäjälle vain tietty määrä kerrallaan. Jotta usean sivun käyttö olisi tehokasta, tulisi datan jakamista hyödyntää jo sitä ladattaessa. Yksinkertaisimmillaan tämä tarkoittaa sitä, että dataa ladataan sitä mukaa, kun käyttäjä sitä tarvitsee. Näin ohjelman käyttö nopeutuu ja suorittimen kuorma kevenee. (5.)

Yksi yleisimmistä käyttöliittymätekniikoista on valintaikkuna (dialog box) eli ikkuna, johon käyttäjältä odotetaan vastausta. Tämä saattaa osoittautua ongelmaksi MVVM-mallissa, sillä Silverlightissa kaiken koodin tulee olla asynkronista. Helpoin ratkaisu tähän ongelmaan on luoda dialogi käyttöliittymän takana ja kytkeä vastaukseen (response) takaisinkutsu (callback). View model pystyy tällöin jonkin tilan (state) tai käskyn muutoksen perusteella käynnistämään dialogipalvelun (dialog service). Takaisinkutsu palauttaa vastauksen ja view voi jatkaa prosessin suorittamista normaalisti. (5.)

Asynkronisuus aiheuttaa myös toisen ongelman: Kun tehtävää on aloitettu suorittamaan, mistä tiedetään, milloin tehtävä on saatu tehtyä loppuun? Tyypillisesti tämä saadaan selville rekisteröimällä tapahtuma tehtävän loppumiseen ja datasitomalla lopputulokset. (8.)

Erittäin yleinen ongelma on animaatioiden ja siirtymien toteuttaminen. Tähän on olemassa useita eri ratkaisumalleja. Visual State Aggrogator on menetelmä, jonka avulla animaatio sidotaan käyttöliittymän tapahtumiin ilman, että view modelia käytetään animaatioon lainkaan. (9.) Jos view modelia halutaan käyttää, voi ratkaisuna käyttää animaation delegaattia (animation delegate) (10), käänteistä ICommandia (reverse ICommand) (11) tai VisualStateManageria (12).

Globaalit muuttujat ja konfigurointitiedot voivat aiheuttaa kehittäjille päänvaivaa. Useimmissa tapauksissa konfiguroinnin voi esittää käyttöliittymässä (IConfiguration), johon konfiguraatiotiedot implementoidaan. Jos view model tarvitsee näitä tietoja, ei sen tarvitse kuin tuoda (import) kyseinen implementaatio. (5.)

2.3 Model-View-Presenter

Model-View-Presenter (MVP) on MVVM-mallin kanssa samankaltainen kehitysmenetelmä, joka esiteltiin vuonna 1996 (5). Molemmissa menetelmissä pyritään erottamaan model ja view erillisiksi osikseen, joiden välillä on toimintaa ohjaava logiikka. MVP-mallissa tätä välikappaletta kutsutaan nimellä presenter. MVP-mallia käytetään erityisesti WinForms-maailmassa. (7.)

MVVM-mallista poiketen MVP käyttää IView-rajapintaa, jonka avulla view saa näytettävät kohteensa. Tämä johtuu siitä, ettei MVP-mallissa käytetä datasidoksia. View kommunikoi suoraan presenterin kanssa kutsumalla sen funktioita. View onkin täysin tietoinen presenterin olemassaolosta. Näin ollen käyttöliittymä ja toimintakoodi sijaitsevat eri tiedostoissa, mutta ne kommunikoivat suoraan keskenään. MVP-mallia käytetäänkin tilanteissa, jolloin sidoksen muodostaminen datakontekstiin (datacontext) ei ole mahdollista, kuten Windows Forms -ohjelmassa. (7.)

2.4 Model-View-Controller

Model-View-Controller (MVC) on yksi vanhimmista Windows-tekniikoista, sillä se kehitettiin jo vuonna 1979 (5). Myös MVC erottaa view’n ja modelin erilleen, mutta niiden välillä toimiva välikappale on nimeltään controller. MVC on käytössä muun muassa ASP.NET-kehityksessä. (7.)

MVC-mallissa view lähettää controllerille tietoa takaisinkutsun tai rekisteröidyn käsittelijän (registered handler) avulla. Esimerkiksi Internet-sivustolla view lähettää controllerille tapahtumat URL:n välityksellä, joka reititetään oikealle controllerille ja metodille. MVC-menetelmässä view voidaan päivittää suoraan modelista ilman, että tiedon tarvitsee kulkea controllerin läpi. MVC-menetelmää tulisi käyttää silloin, kun view’n ja muun ohjelman välillä ei ole selvää yhteyttä. ASP.NET-sivusto on tästä hyvä esimerkki, sillä siinä käyttöliittymä ja logiikka ovat toisistaan täysin erotettuja. (7.)

2.5 Portable Class Libraries

Portable Class Libraries (PCL) on luokkakirjasto, jonka avulla koodin pystyy hyödyntämään Windows 8-, Windows Phone-, Xbox 360- ja muiden .NET-alustojen yhtäaikaisessa kehityksessä. Ilman PCL-kirjastoa koodin joutuisi kirjoittamaan kaikille alustoille erikseen. PCL löytyy valmiiksi asennettuna Visual Studio 2012 -kehitystyökalusta, ja Visual Studio 2010:ssä sen saa toimimaan erikseen projektiin lisäämällä. (13.)

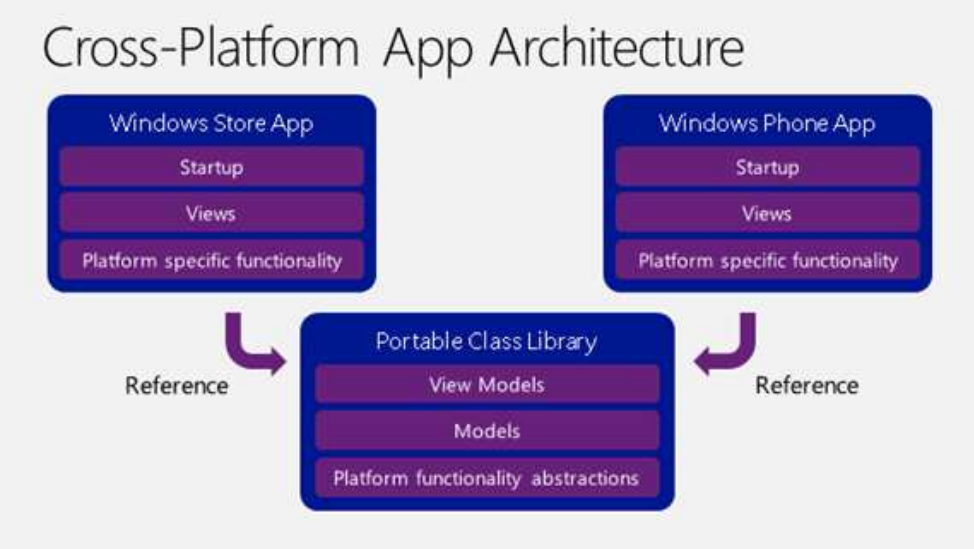
Koska kaikki ominaisuudet eivät ole käytössä eri alustoilla, aiheuttaa tämä tiettyjä rajoituksia PCL:n käyttöön. Esimerkiksi ThreadStaticAttribute, jonka avulla staattiset kentät (static fields) voivat saada jokaisessa säikeessä (thread) uniikin arvon, ei ole tuettu Windows Phone- ja Xbox-alustoilla. Mitä useampaa alustaa haluaa hyödyntää, sitä vähemmän ominaisuuksia projektissa voi käyttää. Kuvassa 2 on esitetty tiivistelmä eri alustojen ominaisuuksista. (14.)

Feature	Silverlight		Windows Phone		Metro style apps	.NET Framework			Xbox 360
	4	5	7	7.5		4	4.0.3	4.5	
Core BCL	✓	✓	✓	✓	✓	✓	✓	✓	✓
Core XML	✓	✓	✓	✓	✓	✓	✓	✓	✓
LINQ	✓	✓	✓	✓	✓	✓	✓	✓	
IQueryable	✓	✓		✓	✓	✓	✓	✓	
`dynamic` keyword	✓	✓			✓			✓	
Core WCF	✓	✓	✓	✓	✓	✓	✓	✓	
Core networking	✓	✓	✓	✓	✓	✓	✓	✓	
View models	✓	✓	✓	✓	✓			✓	
Data annotations	✓	✓			✓		✓	✓	
XLINQ	✓	✓	✓	✓	✓		✓	✓	✓
MEF	✓	✓			✓	✓	✓	✓	
Data contract serialization	✓	✓	✓	✓	✓	✓	✓	✓	
XML serialization	✓	✓	✓	✓	✓	✓	✓	✓	
JSon serialization	✓	✓	✓	✓	✓	✓	✓	✓	
System.Numerics	✓	✓			✓	✓	✓	✓	

KUVA 2. PCL-ominaisuuksien tuki eri alustoilla (14)

PCL:n hyötykäytöllä voidaan MVVM-menetelmässä käyttää jokaisella alustalla samoja modeleja ja view modeleja. Eri alustoille tarvitsee siten ohjelmoida ai-

noastaan oma käynnistymismekanismi, omat view't ja alustalle ominaiset ominaispiirteet (kuva 3). (14.)



KUVA 3. Usean alustan arkkitehtuuri PCL:ää käytettäessä (14)

Käytännössä PCL:ää hyödynnettäessä luodaan kokonaisuudesta yksi solution, johon sisältyvät PCL-osuus yhtenä projektina ja kaikkien eri alustojen osuudet omina projekteinaan. Kuhunkin eri alustan projektiin lisätään referenssi PCL-projektiin, jonka jälkeen datakonteksti datasidotaan tähän referenssiin. (15.)

2.6 Managed Extensibility Framework

Managed Extensibility Framework (MEF) on kirjasto, jonka avulla kehittäjät voivat tehdä ohjelmiin laajennuksia ilman erillisen konfiguroinnin tarvetta. Jokainen MEF:ää käyttävä ohjelma tutkii automaattisesti laajennuksien olemassaolon, jolloin uusien ominaisuuksien luominen ohjelmiin on helppoa. MEF on integroitu .NET Framework 4:ään, ja se on käytössä kaikissa .NET Frameworkia tukevissa sovelluksissa. (16.)

Esimerkkinä MEF:n hyödyntämisestä voidaan käyttää yksinkertaista laskukonetta. Laskukone käy läpi kaikki määritellyssä kansiossa olevat laajennukset ja etsii sieltä annetuilla parametreilla käytettävän metodin. Laskukoneessa metodeja voivat olla vaikkapa yhteenlasku, vähennyslasku ja jakolasku. (16.)

3 KOHDEALUSTAT

Projekti toteutettiin lähtökohtaisesti kolmelle alustalle: Windows 8 -pöytäkoneelle, Windows Phone 7 -puhelimelle sekä Windows RT -tabletille. Alustojen erilaisuus vaikutti voimakkaasti tuotteen suunnitteluvaiheeseen, sillä tuotteen ominaisuudet eivät välttämättä toimi sellaisenaan eri alustoilla.

3.1 Windows 8

Windows 8 on Microsoftin kehittämä käyttöjärjestelmä, joka julkaistiin 26.8.2012 (17). Käyttöjärjestelmän kehitys aloitettiin jo, ennen kuin sen edeltäjä Windows 7 oli julkaistu (18).

Windows 8 on monella tapaa erilainen kuin edelliset Windowsin versiot. Selkein eroista on uusi aloitusnäkymä, joka korvaa entisen Käynnistä-valikon. Aloitusnäkymä ei ole edellisten versioiden Käynnistä-valikon tapaan ruudun vasemmassa ala-nurkassa, vaan se täyttää koko näytön. (19.) Eri ohjelmat esitetään näkymässä tapahtumaruutuina, joista painamalla ohjelmia voidaan suorittaa. (Kuva 4.) Ruuduista pystyy myös näkemään nopeasti tietoja, kuten saapuneiden sähköpostien määrän. Minkä tahansa ohjelman tai kansion pystyy lisäämään ruuduksi aloitusnäkymään. (20.) Uusi aloitusnäkymä on saanut paljon kritiikkiä käyttäjiltä, ja vanhan Käynnistä-valikon korvaavia ohjelmia on ilmestynyt kolmannen osapuolen kehittäjiltä useita (21). Aloitusnäkymä (ja ylipäätään koko Windows 8) toimii parhaiten kosketusnäytöllä (22).

KUVA 4. Windows 8:n aloitusnäkymä (23)

Windows 8 tukee useiden näyttöjen käyttöä paremmin kuin edelliset versiot. Yhdellä näytöllä voi pitää aloitusnäkymää ja toisella työpöytänäkymää tai vaihtoehtoisesti molemmilla näytöillä voi näyttää työpöytänäkymän. Kaikille näytöille voi asettaa erilaisen taustakuvan toisin kuin edellisissä versioissa. Lisäksi sovelluksissa voi käyttää jaettua näyttöä, jolloin näyttötila jaetaan kahden sovelluksen kesken. (19.)

Pilvipalvelut ovat tärkeässä osassa Windows 8:ssa. Microsoftin SkyDrive-ominaisuudella tiedostoja voi jakaa verkon kautta eri laitteille. SkyDrive-ominaisuus löytyy myös muun muassa Office 2013 -työkalusta. (19.)

Windows 8 on selvästi nopeampi kuin edelliset Windowsin versiot. Esimerkiksi käynnistyminen tapahtuu Windows 8:lla puolet nopeammin kuin Windows 7:llä. Myös tiedostojen siirto on nopeampaa ja pelien ruudunpäivitys parempi. (22.)

3.2 Windows Phone 7

Windows Phone 7 on Microsoftin kehittämä mobiilikäyttöjärjestelmä, joka korvasi aikaisemmat Windows Mobile -laitteet (23). Se julkaistiin Euroopassa 21.8.2010 (24).

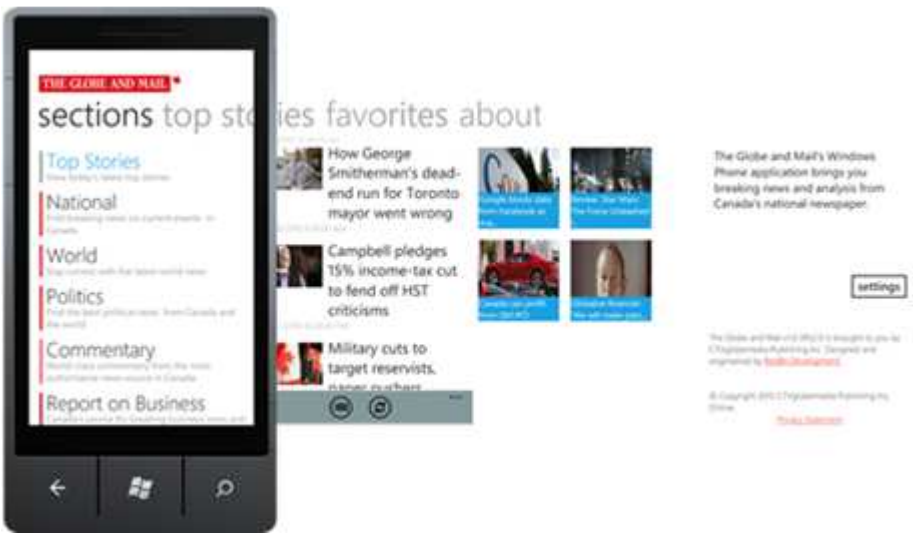
Microsoft on määrittänyt laitevalmistajille tarkat määritykset siitä, mitkä vaatimukset Windows Phone 7:n tulee täyttää: Ainoastaan 800 x 480 -tarkkuuden kosketusnäytöt kelpaavat. Laitteessa on oltava viisi fyysistä nappia, joiden toiminta on aina samanlainen (paluunappi, aloitusnappi, Bing-hakukonenappi, kameranappi ja virtanappi). Kameran on yllettävä vähintään 5 megapikselin tarkkuuteen. RAM-muistia on oltava vähintään 256 MB ja kiintolevyn on oltava vähintään 8 GB:n kokoinen. Jokaisessa laitteessa on oltava WiFi, AGPS, kiihtyvyysanturi sekä FM-radio. (27.)

Itse käyttöjärjestelmä koostuu samankaltaisista tapahtumaruuduista kuin Windows 8. Käyttöjärjestelmä on pyritty pitämään mahdollisimman yksinkertaisena poistamalla muun muassa turhat valikot ja resurssienhallinta. (27.) Jotta käyttökokemus pysyisi yksinkertaisena, on Microsoft tehnyt myös sovelluksille määritykset. Esimerkiksi fyysisten nappien toiminnan on oltava kaikissa sovelluksissa sama. Kaikkien ohjelmien on käynnistyttävä viidessä sekunnissa ja oltava käyt-

tövalmiita 20 sekunnissa. (28.) Kaikki Windows Phone Marketplacella julkaistavat tuotteet käyvät läpi Microsoftin sertifiointiprosessin, jossa tuotteen toiminta testataan (29).

Käyttöjärjestelmän teemaksi voi valita joko vaalean tai tumman teeman. Tämä vaikuttaa käytännössä siihen, onko käytössä musta teksti vaalealla taustalla vai valkoinen teksti mustalla taustalla. Lisäksi puhelimen korostusvärin (eli tapahtumaruutujen ja otsikoiden värin) voi vaihtaa. (27.)

Sovelluksien tärkeimpiä ominaisuuksia ovat pivot ja panorama, jotka luovat perustan käyttöjärjestelmän käytettävyydelle. Pivot ja panorama ovat sivuja, joiden alasivuja (joita kutsutaan nimellä item) pystyy selaamaan sipaisemalla näyttöä sivuttaissuunnassa. Puhelimesta vakiona löytyvä valikko Ihmiset (eli puhelimen yhteystiedot) on esimerkki panorama-sivusta. Panoraman alasivuja ovat tässä tapauksessa uusimmat viestit, viimeisimmät kontaktit ja kaikki puhelimesta löytyvät yhteystiedot. Pivotin ja panoraman ero on siinä, että pivotissa jokainen alasivu on oma kokonaisuutensa (kuva 5), kun taas panoramassa kaikki sivut luovat yhdessä yhden pitkän kokonaisuuden (kuva 6). Panoramassa yhden sivun leveys voi olla suurempi kuin näytön leveys, mutta pivotissa näin ei ole. (30.)



KUVA 5. Esimerkki pivot-sivustosta (31)

KUVA 6. Esimerkki panorama-sivustosta (31)

Windows Phone 7 -laitteita valmistavat muun muassa ASUS, Dell, HTC, LG ja Samsung (32). Suomalaisille tärkein valmistaja on Nokia, joka siirtyi valmistamaan Windows Phone -puhelimia 11.2.2010 (33). Nokian Windows Phone 7 -puhelimiin eli Lumia-sarjaan kuuluvat muun muassa Lumiat 800 ja 710 (34) sekä isompi ja tehokkaampi 900 (35).

3.3 Windows RT -tabletti

Windows RT on 26.8.2012 julkaistu Microsoftin käyttöjärjestelmä (36), joka on optimoitu litteisiin ja kevyisiin tietokoneisiin. Windows RT -käyttöjärjestelmässä ei voi suorittaa kaikkia Windowsin ohjelmia vaan ainoastaan vakiosovelluksia tai Windows-kaupasta ladattuja sovelluksia. Näin ollen perinteiset työpöytäsovellukset eivät toimi Windows RT:llä. Windows RT on saatavilla tietokoneisiin ja taulutietokoneisiin eli tabletteihin, joissa on ARM-suoritin. (37.)

Windows RT -tabletissa on kosketusnäyttö, jolla sitä ohjataan. Tabletin USB-paikkaan voi kytkeä hiiren, jolloin esimerkiksi käyttöjärjestelmässä vakiona olevan Office-työkalun käyttökokemus paranee huomattavasti. (36.) Esimerkiksi Microsoftin julkaisemaan Surface-tablettiin on mahdollista kytkeä näppäimistö,

jolloin tablettia voi käyttää samaan tapaan kuin tavallista kannettavaa tietokonetta (kuva 7) (38).

KUVA 7. Microsoftin Surface-tabletti näppäimistöön kytkettynä (39)

4 MOBIILITERMINAALIN YKSIKKÖTESTAUS

Built-in self-test (BIST) on testaukseen kehitetty menetelmä, jossa systeemi testaa itse itseään. BIST on yleensä implementoitu itse laitteistoon (40, s. 1), mutta samantapaista tekniikkaa voidaan soveltaa myös ohjelmistokehityksessä (41, s. 3). Yksi ohjelmistokehityksen malli, jossa hyödynnetään automaattista yksikkötestausta, on nimeltään Test-driven development (TDD) (42).

Yksikkötestin perusperiaate on yksinkertainen. Ensin luodaan testi ja määritetään se, mitä tuloksia sillä pitäisi saada aikaan. Sitten luodaan testiä varten koodirunko, joka on niin minimaalinen, että se juuri ja juuri kääntyy ilman virheitä (errors). Kun testi ajetaan, tulee sen epäonnistua tässä vaiheessa, sillä varsinaista testin läpäisevää toiminnallisuutta ei vielä ole. Näin varmistetaan se, että testataan oikeaa ominaisuutta eikä koodi toimi vahingossa. Tämän jälkeen ominaisuus toteutetaan ja varmistetaan, että se läpäisee testin. Jos koodi toimii, voidaan siirtyä seuraavan ominaisuuden toteuttamiseen. Koodia voidaan myös muokata, jotta esimerkiksi mahdolliset duplikaatit saadaan poistettua. Jokaisen muutoksen jälkeen kaikki testit ajetaan uudelleen, jotta varmistetaan ominaisuuksien toimivuus. (42.) TDD-malli voidaan yksinkertaistaa ajatukseen Punainen–Vihreä–Muokkaus (Red–Green–Refactor) eli epäonnistuminen (punainen), toimiva koodi (vihreä) ja koodin mahdollinen parantaminen (muokkaus) (kuva 8) (43).

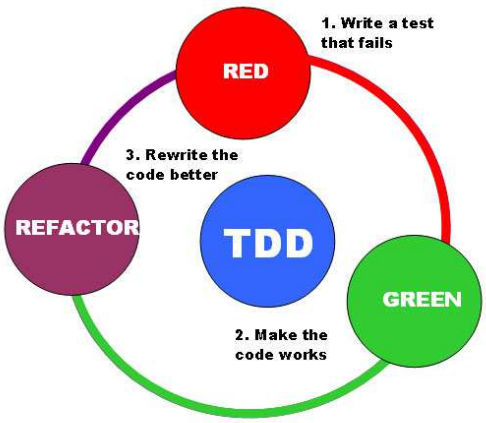


Figure 1: The mantra of Test-Driven Development (TDD) is "red, green, refactor".

KUVA 8. TDD-mallin kolme vaihetta (43)

Hyvän yksikkötestin on toimittava nopeasti. Esimerkiksi tietokannat ja verkkoyhteydet testataan erikseen tai simuloidaan, sillä ne hidastavat testausta eikä niitä käyttämällä saada varmaa tietoa siitä, missä ongelma vikatilanteessa oikeasti on. Yhden testin on myös testattava vain yhtä asiaa, jotta tiedetään varmasti, missä vika on. Testien on toimittava kaikissa ympäristöissä. Lisäksi testien on oltava ymmärrettäviä eli toisen kehittäjän on ymmärrettävä, mikä testin tarkoitus on. (42.)

Yksikkötestauksella on lukuisia etuja. Joukko yksikkötestejä antaa jatkuvaa palautetta siitä, että kaikki komponentit toimivat oikein. Yksikkötestit eivät myöskään vanhene toisin kuin pelkkä testausdokumentaatio. Kun testi saadaan suoritettua ja koodiduplikaatit poistettua, tiedetään varmasti koodin olevan valmis ja ohjelmoija voi siirtyä seuraavan ominaisuuden pariin. Yksikkötestaus korostaa myös suunnittelun tärkeyttä, sillä ohjelmoija ei voi toteuttaa ominaisuutta, jos hän ei tiedä, mitä siitä pitää testata. Koodia voidaan myös muokata milloin tahansa, sillä testien ansiosta muutosten toimivuus tiedetään välittömästi. Myös virheiden (bugs) löytäminen helpottuu: jos virhe löytyy, tekee ohjelmoija testin, joka paljastaa sen, ja muuttaa koodia siten, että virhe katoaa ja kaikki testit saadaan edelleen hyväksytysti suoritettua. (42.)

5 TEKNIIKOIDEN SOVELTAMINEN

5.1 Mobiilin toiminnanohjausjärjestelmän erityisvaatimukset

Mobiilin toiminnanohjausjärjestelmän toimintaperiaatteita tutkittaessa löydettiin seuraavat erityisvaatimukset:

1. Ohjelman eri versioista on löydyttävä samat perusominaisuudet, mutta laitteilla voi olla alustariippuvaisia erityistoimintoja.
2. Testaus on kyettävä suorittamaan kaikilla alustoilla samalla tavalla.
3. Päätelaitteen on haettava tietokannasta vain ne tiedot, jotka ovat muuttuneet edellisen latauskerran jälkeen.
4. Ohjelman on toimittava, vaikka yhteys palvelimeen katkeisikin.

Eri versioiden perusominaisuudet

Projektin monialustaisuus aiheuttaa tiettyjä erityishaasteita. Koodin tulee olla sellaista, että sitä voidaan hyödyntää sellaisenaan mahdollisimman paljon eri alustoilla. Alustojen erilaisuus on kuitenkin otettava huomioon esimerkiksi käyttöliittymiä suunnitellessa. Ohjelman eri versioista on löydyttävä samat perusominaisuudet, mutta laitteilla voi olla joitain alustariippuvaisia erityistoimintoja. Ohjelma halutaan myös pitää sellaisena, että yhtä versiota käytettyään käyttäjä osaa käyttää myös muita versioita välittömästi. (44.)

Testaus

Koska projekti toteutetaan usealle alustalle yhtä aikaa, on sen testaamiseen kiinnitettävä erityistä huomiota. Testien on oltava sellaisia, että niitä voidaan soveltaa kaikille alustoille samalla tavalla. Testaukseen onkin tehtävä erillinen käyttöliittymä, joka voidaan yhdistää kaikkien alustojen view modeleihin. (45.)

Tietokantayhteys

Ohjelman toimintaan kuuluu oleellisesti se, että kaikilla asiakkaan käyttäjillä on yksi yhteinen tietokanta, jota käytetään serverin välityksellä. Päätelaitteen kytkeytyessä palvelimeen ladataan laitteen muistiin kaikki tarvittavat tiedot. Tämän jälkeen haetaan ainoastaan ne tiedot, jotka ovat muuttuneet edellisen latausker-

ran jälkeen. Jos jokaisen latauksen yhteydessä haettaisiin aina kaikki tiedot, hidastuisi ohjelman käyttö merkittävästi. (46.)

Offline-tila

Toiminnanohjausjärjestelmän on tarkoitus toimia, vaikka yhteys palvelimeen onkin katkennut. Tätä toiminnallisuutta kutsutaan WiseMaster-tuotteissa offline-tilaksi. Käytännössä muokatut tiedot tallennetaan päätelaitteen muistiin ja lähetetään palvelimelle, kun yhteys on jälleen saatavilla. (46.)

5.2 Tekniikoiden soveltamisohjeet

Mobiilin toiminnanohjausjärjestelmän erityisvaatimuksista ja Windows-tekniikoista muodostettiin seuraavat soveltamisohjeet:

1. Projektissa tulee käyttää Model-View-ViewModel (MVVM) -kehitysmallia yhdessä Portable Class Libraries (PCL) -luokkakirjaston kanssa.
2. Testauksessa tulee hyödyntää Test-driven development (TDD) -mallia.
3. Ohjelman suunnittelu tulee aloittaa käyttöliittymästä.
4. Käyttöliittymässä tulee kiinnittää erityistä huomiota erikokoisiin näyttötiloihin.
5. Serverin on toimittava kaikilla alustoilla samalla tavalla.
6. Tietokannassa käytetään aikaleimaa, jonka avulla päätelaitteelle saadaan ladattua vain muuttuneet tiedot.
7. Offline-tila on huomioitava ohjelman koko kehityskaaren ajan.

MVVM-malli ja PCL-luokkakirjasto

Kaikki tutkitut Windows-tekniikat eivät soveltuneet kohdeprojekteihin sellaisenaan. Esimerkiksi Model-View-ViewModel (MVVM), Model-View-Presenter (MVP) ja Model-View-Controller (MVC) käytännössä kumoavat toisensa, joten niistä vain yhtä voitiin käyttää. MVVM-mallin todettiin tukevan parhaiten mobiilia toiminnanohjausjärjestelmää. MVVM-mallia tukemaan päätettiin ottaa Portable Class Libraries (PCL) -luokkakirjasto. Tätä mallia tukee monialustaisen tuotekehityksen helppous sekä näkymien, toimintojen ja datatyyppien jakaminen erilleen, jolloin eri ohjelmoijat voivat toteuttaa eri osioita samanaikaisesti. Lisäksi ohjelmien toimintojen ollessa samanlaisia on koodin uudelleenkäytön maksi-

mointi suotavaa, ja PCL on juuri omiaan tähän tehtävään. Monen alustan tuotekehitys helpottuukin huomattavasti MVVM-mallin ja PCL-kirjaston yhteiskäytöllä. Managed Extensibility Framework (MEF) vaikutti tekniikkana mielenkiintoiselta, mutta sille ei kohdeprojekteissa ollut käyttöä. MEF kannattaa kuitenkin muistaa suunniteltaessa tulevia projekteja. (47.)

Testaus

Projektissa tulisi soveltaa Test-driven development -mallia. Kukin ohjelman toiminnallisuus kirjataan ylös esimerkiksi PowerPoint-dokumenttiin (yksi ominaisuus/dia), johon kirjataan ylös ominaisuudelle lähetettävä data, ominaisuuden toiminnallisuus ja ominaisuuden palauttama data. Kukin yksikkötesti testaa aina yhtä ominaisuutta kerrallaan, ja testien perusteella voidaan varmistaa ominaisuuksien haluttu toimivuus. Testissä käytetyt datat säilytetään erillisessä tiedostossa, jolla korvataan oikeassa ympäristössä käytettävät datayhteydet. Kuhunkin ohjelman ominaisuuteen ohjelmoidaan erillinen testitila, jonka avulla testit saadaan suoritettua normaalin toiminnallisuuden sijaan. (45.)

Käyttöliittymä projektin aloituskohtana

Ohjelman suunnittelussa käyttöliittymä on ensisijaisen tärkeä. Ohjelman halutaan olevan kaikilla alustoilla mahdollisimman samankaltainen, jotta käyttäjä osaisi käyttää sen eri versioita mahdollisimman saumattomasti. Niinpä jos käyttöliittymä suunnitellaan yhdelle alustalle huolellisesti, voidaan se siirtää toiselle alustalle huomattavasti helpommin. Käyttöliittymä onkin hyvä aloituskohta projektille. Aluksi käyttöliittymä suunnitellaan ilman mitään toiminnallisuuksia. Toiminnot lisätään vasta, kun kaikki käyttöliittymän elementit ovat valmiita. Kun käyttöliittymä on valmis yhdellä alustalla, voidaan sen toiminnallisuutta alkaa toteuttaa samalla, kun käyttöliittymä siirretään muille alustoille. (47.)

Erikokoisten näyttökokojen huomiointi käyttöliittymässä

Käyttöliittymässä on kiinnitettävä erityistä huomiota erikokoisiin näyttötiloihin. Puhelinta voidaan pitää pysty- tai vaakasuorassa, Windows 8 -sovellusta vaakasuorassa tai supistetussa tilassa ja tablettisovellusta pystysuorassa, vaakasuorassa tai supistetussa tilassa. Toimintojen on oltava käytössä ja käyttöliittymän selkeä käytetystä näyttötilasta riippumatta. (47.)

Serveri

Serveri on myös olennainen osa projektia. Serverin tulee olla sellainen, että kaikki eri alustat voivat ottaa siihen yhteyden ja kommunikoida sen kanssa samalla tavalla. Serverin toiminnot testataan yksikkötestillä mahdollisimman varhaisessa vaiheessa. (47.)

Tietokannan aikaleima

Päätelaitteen on osattava hakea vain ne tiedot, jotka ovat muuttuneet edellisen hakukerran jälkeen. Tämä on ratkaistavissa erillisellä aikaleimalla, joka lisätään jokaiseen tietokannan tauluun. Kun jotain tietoa muokataan, päivitetään tiedon aikaleimakenttä. Päätelaitteelle tallennetaan aikaleima aina, kun tietoja haetaan palvelimelta. Palvelin palauttaa vain ne tiedot, jotka ovat muuttuneet päätelaitteen aikaleiman merkitsemisen jälkeen. (46.)

Offline-tila

Offline-tila on huomioitava ohjelmaa suunniteltaessa. Offline-tilaan siirrytään, jos yhteys palvelimeen menetetään. Offline-tilaan siirtymisestä on annettava käyttäjälle selkeä palaute, jotta hän tietää tietojen olevan vain paikallisessa muistissa. Vaatimuksena offline-tilan käytölle on, että käytettävät datat on tallennettu päätelaitteen muistiin. Ohjelmalla on offline-tilassa pystyttävä käyttämään sen tärkeimpiä toimintoja samalla tavalla kuin online-tilassa. Tiedot tallennetaan päätelaitteen muistiin, josta ne lähetetään palvelimelle yhteyden palattua. Tämä voi kuitenkin aiheuttaa konfliktin, jos muut käyttäjät ovat muokanneet samoja tietoja. Onkin tärkeää huomioida, miten konfliktitilanteessa toimitaan. Yksi vaihtoehto on toimia siten, että palvelimelle tallennetaan tiedot sitä mukaa, kun ne sinne saapuvat. Tämä aiheuttaa sen, että ainoastaan viimeisenä palvelimeen yhteyden saavan henkilön tiedot tallentuvat konfliktitilanteessa. Toinen vaihtoehto on ilmoittaa käyttäjälle, että tiedot ovat muuttuneet sen jälkeen, kun yhteys palvelimeen menetettiin. Tämän jälkeen hän voi valita, mitkä tiedot tallennetaan palvelimelle. Kolmantena vaihtoehtona lähetetään konfliktista erillinen tieto ylläpitäjälle, joka voi valita lopullisesti tallennettavat tiedot. Tässä tapauksessa on kuitenkin päätettävä, mitä tietoja käyttäjä käsittelee ylläpitäjän ratkaisua odotettaessa. (46.)

5.3 Tekniikoiden implementointi kohdeprojekteihin

Soveltamisohjeita sovellettiin WiseMaster Myynti -tuotteen Windows Phone 7-, Windows RT -tabletti- ja Windows 8 -versioissa. Soveltamisohjeiden negaationa käytettiin WiseMaster Huolto -tuotteen Windows RT -tablettiversion kehitystyötä, jossa ei käytetty MVVM- eikä PCL-tekniikoita. Koska kyseessä on edelleen kehityksessä olevat tuotteet, ei ohjelmista ole esitetty tässä ruudunkaappauksia.

WiseMaster Huollon Windows RT -tablettiversio toteutettiin käyttäen pohjana aikaisempaa Windows Phone 7 -versiota, jossa ei ollut käytetty MVVM-, MVC- tai MVP-arkkitehtuuria. Ohjelmasta haluttiin hyödyntää mahdollisimman paljon olemassa olevaa koodia resurssien säästämiseksi ja turhan työn minimoimiseksi. Haasteeksi muodostui erityisesti käyttöliittymän muokkaus, sillä käyttöliittymä oli tehty puhtaasti puhelimen ehdoilla. Ratkaisuna hyödynnettiin monin paikoin Split Page -mallia eli näkymää, jossa näytön vasemmalla puolella näytettiin lista tiedoista, kuten asiakkaista ja työmääräyksistä, kun taas näytön oikealla puolella näytettiin lisätietoja valitusta kohteesta. Näin pystyttiin hyödyntämään paljon vanhoja näkymiä ilman, että ne olisivat näyttäneet tabletin näytöllä liian suurilta tai yksinkertaisilta.

WiseMaster Myynti aloitettiin Windows Phone 7 -version käyttöliittymän kehittämisellä. Käyttöliittymä toteutettiin ilman toiminnallisuuksia hyödyntäen väliaikaista dataa muun muassa listojen suunnittelussa. Samanaikaisesti toteutettiin serverin toimintoja ja dataluokkia. Kun Windows Phone 7 -version käyttöliittymä valmistui, aloitettiin Windows RT -tabletin käyttöliittymän suunnittelu. Windows 8 -versio syntyi tablettiversion kanssa samanaikaisesti, sillä koodi pystyttiin hyödyntämään näiden versioiden kanssa sellaisenaan.

Kehitettäessä Windows 8- ja erityisesti Windows RT -versioita oli otettava huomioon erilaisten näkymien mahdollisuudet. Windows 8 mahdollistaa kahden eri ohjelman käytön yhtä aikaa siten, että toinen ohjelmista käyttää vain 320 pikselin levyistä osuutta näytöstä. Lisäksi tabletin pystyy kääntämään myös pystysuuntaan. Eri näkymien käytöt hoidettiin säätämällä objektien, kuten tekstilaatikoiden, kokoa ja sijaintia käytetyn näkymän mukaan. Lisäksi esimerkiksi osa listoista käytti kokonaan erilaista muotoilua erikokoisissa näkymissä. Osa nä-

kymistä toteutettiin siten, että laitteen ollessa vaakasuorassa käyttäjälle näytettiin kaikki data eri sarakkeissa, mutta käytettäessä pystysuoraa tai supistettua näkymää data jaettiin omiin välilehtiinsä.

WiseMaster Myyntiä kehitettäessä tarkoituksena oli käyttää eri versioille yhteistä WMCore-kirjastoa, joka toteutettaisiin PCL-tekniikkaa hyödyntäen. Windows RT -version kehitystä aloitettaessa todettiin kuitenkin, että käytetty tiedostotyyppi olikin ollut Phone Class Library eikä Portable Class Library. Erehdys ilmeisesti tapahtui siksi, että Windows Phone 7 -kehittäjällä oli käytössään Visual Studio 2010, eikä siinä ole Portable Class Libraryä valmiiksi asennettuna. WMCore jouduttiin erehdyksen takia kuitenkin kirjoittamaan kokonaan uudestaan, sillä siinä oli käytetty tabletissa toimimattomia tekniikoita, kuten Windows.Phone-luokkakirjastoa. MVVM-tekniikka osoittautui kuitenkin sikäli hyödylliseksi tässä tilanteessa, että tablettiversion näkymiä voitiin toteuttaa, vaikkei toiminnallisuuksia ollut käytössä WMCore-tiedoston puuttumisen takia.

6 TOTEUTUKSEN ARVIOINTI

WiseMaster Huolto saatiin siirrettyä nopeasti tablettiversioksi. Huollosta oli olemassa Windows Phone 7 -versio, mikä nopeutti tuotteen kehitystä huomattavasti verrattuna WiseMaster Myynti -tuotteeseen, josta oli olemassa vain Windows Mobile -versio. Huollon kehitystä edesauttoi myös se, että se käytti uudempaa ja selkeämpää tietokantarakennetta ja sen serveriä voitiin käyttää täysin sellaisenaan. Huollon kehityksessä todettiin myös määrittelyvaiheen tärkeys, sillä kun vaatimusluettelot olivat valmiita ja huolellisesti laadittuja, helpottui ohjelmointivaihe merkittävästi.

WiseMaster Myynti -tuotteen kehityksessä oli joitakin varsin opettavaisia ongelmia. Muut projektit veivät resursseja Myynnin kehityksestä, minkä takia ominaisuuksia jouduttiin monin paikoin karsimaan. Esimerkiksi alkuperäisen suunnitelman mukainen serverin TDD-mallia noudattava yksikkötestaus jouduttiin jättämään pois, ja serveriä testattiin muilla tavoin. Varsinaisen yksikkötestauksen puute aiheutti kuitenkin sen, että serverissä oli yhteysongelmia varsinkin Windows 8 -ympäristössä. Tämä hankaloitti view modelien ohjelmointia mobiiliterminaalissa.

Toinen testauksen puutteen aiheuttama viivästys oli WMCore-kirjaston väärä tiedostomuoto. Koska WMCore sisälsi suuren osan ohjelman toiminnoista, ei tablettiversiosta voinut toteuttaa kuin käyttöliittymän eli view’t ennen WMCoren uudelleenohjelmointia. Tapauksesta kuitenkin opittiin se, että kehitettäessä monelle alustalle ominaisuuksia on ensiarvoisen tärkeää testata eri komponenttien yhteensopivuutta mahdollisimman aikaisessa vaiheessa.

Myynnin kehitystä hidasti myös riittävän laajojen vaatimusluetteloiden puute. Joitakin Windows Mobile -version ominaisuuksia ei ollut järkevä tuoda sellaisenaan uusiin versioihin, ja osa ominaisuuksista jätettiin kokonaan pois. Kuitenkaan kaikilla kehittäjillä ei ollut varmuutta siitä, mitkä ominaisuudet lopulliseen tuotteeseen otetaan mukaan, mikä puolestaan jarrutti tuotteen kehitystä. Jos heti alussa olisi tehty selkeät luettelot mukaan otettavista ominaisuuksista, olisi kehitystyö ollut huomattavasti sujuvampaa. Tämä osoittautui todeksi serverin

kehityksessä. Serverin toteutus aloitettiin sillä, että eri toiminnot ja datatyypit kirjattiin ylös ja niiden oikeellisuus varmistettiin yhteisissä palavereissa. Tämä helpotti huomattavasti sekä serveriohjelmoijan että käyttöliittymäsuunnittelijan työtä.

WiseMaster Huollossa ei käytetty MVVM-mallia, mutta Myynnissä sitä käytettiin. Huollon kehityksessä MVVM-mallia ei juurikaan osannut kaivata, mutta Myynnissä se osoittautui hyödylliseksi. MVVM-mallin hyödyllisyys riippuukin vahvasti projektin luonteesta. Ensinnäkin Huollon kehityksessä käytettiin vain yhtä alustaa. Tämän takia ei ollut perusteltua käyttää PCL-kirjastoa. Toiseksi Myyntiä kehitti yhtä aikaa useita henkilöitä, mutta Huollon tablettiversiota toteutti pääasiassa yksi ohjelmoitsija. Niinpä töitä ei ollut tarpeen jakaa esimerkiksi käyttöliittymään ja toiminnallisuuksiin, vaan ohjelmaa voitiin toteuttaa toiminto kerrallaan. Ainoa kerta, kun MVVM-mallista olisi Huollon kehityksessä ollut hyötyä, oli työmääräykseen lisättyjen tuotteiden muokkaus. MVVM-mallissa dataa olisi voinut käsitellä yhdellä view modelilla, mutta nyt dataa joutui käsittelemään useissa eri paikoissa. Kuitenkaan Huollon kaltaisissa pienemmissä projekteissa MVVM-malli ei tuo niin paljoa hyötyä, että sitä kannattaisi niissä soveltaa. Sen sijaan Myynti hyötyi selvästi MVVM-mallin käytöstä.

PCL-luokkakirjastoa käytettäessä pitää olla tarkkana käytettyjen dll-tiedostojen kanssa. Esimerkiksi projektissa käytetystä MVVM Light -pohjasta on käytettävä eri tiedostoa riippuen siitä, mitä alustoja projektissa on mukana. PCL osoittautui yhdessä MVVM-mallin kanssa toimivaksi ratkaisuksi, jonka ansiosta esimerkiksi view modelit ja luokkarakenteet pystyi hyödyntämään sellaisenaan eri alustoilla. Usean alustan tuotekehityksessä kannattaakin ehdottomasti käyttää MVVM-mallin kanssa PCL-kirjastoa.

PCL-kirjastoa käyttöönottaessa ongelmia tuottivat lähinnä navigointi ja puhelimelle tyypilliset ominaisuudet, kuten soittaminen ja viestin näyttäminen dialogissa. Nämä ongelmat saatiin kuitenkin ratkaistua melko nopeasti, ja ratkaisujen löydyttyä ohjelmointityö nopeutui selvästi. Navigointi ratkaistiin muodostamalla PCL-kirjastoon perusluokka, jonka kunkin alustan projekti perii. Näin kullakin alustalla voidaan käyttää alustalle ominaista navigointiratkaisua, vaikka view

model huolehtiikin navigoinnin suorittamisesta. Puhelimen ominaisuudet toteutettiin tavalliseen tapaan käskyjen (command) avulla, mutta näissä ominaisuuksissa view model palauttaa puhelinprojektille vastauksen. Vastauksen saatuaan puhelinprojektissa suoritetaan haluttu toiminto, kuten soittotapahtuma.

Käyttöliittymään panostaminen oli kannattava ratkaisu, ja ohjelmien käyttö todettiin varsin miellyttäväksi. Puhelimen ja tabletin näkymät muistuttivat paljon toisiaan, mutta tabletissa oli kuitenkin huomioitu näytön isompi koko. Eri näyttötilat oli myös otettu suunnittelussa huomioon, ja ohjelmat tuntuivat varsin käyttökelpoiselta kaikissa tiloissa.

Offline-tila päätettiin toteuttaa siten, että viimeisin päivitys tallennetaan tietokantaan. Vanhat tiedot kuitenkin tallennetaan erilliseen historiakantaan, josta ne voidaan tarpeen mukaan palauttaa. Tämä järjestely todettiin helpoimmaksi käyttäjän kannalta, sillä pidemmän offline-jakson jälkeen olisi todella tympeää joutua hyväksymään jokainen muutos erikseen. Ylläpitäjän hyväksyntä olisi ollut ratkaisuna liian monimutkainen, sillä pahimmillaan konfliktin aiheuttanut kohde voisi olla konfliktitilassa hyvinkin pitkään ylläpitäjän hyväksyntää odotettaessa.

Kaiken kaikkiaan soveltamisohjeet vaikuttivat olevan varsin päteviä. Ainoastaan ohjeissa mainittu TDD-mallin hyödyntäminen jäi kohdeprojekteissa kokeilematta, mutta siitäkin olisi ollut hyötyä. Mallia kannattaakin testata tulevissa projekteissa. Soveltamisohjeita ei kannata tulevaisuudessa unohtaa, sillä niiden oikea käyttö edesauttaa tuotekehitystä merkittävästi.

7 YHTEENVETO

Työn tavoitteena oli tutustua erilaisiin Windows-tekniikoihin ja laatia niiden pohjalta soveltamisohjeet, joiden avulla niitä voidaan hyödyntää useiden alustojen tuotekehityksessä. Soveltamisohjeita testattiin käytännössä WiseMaster Huolto- ja WiseMaster Myynti -tuotteiden tuotekehityksessä.

Työn aikana tutustuttiin eri Windows-tekniikoihin, joista valittiin usean alustan tuotekehityksen kannalta oleellisimmat. Yhdistämällä nämä tekniikat mobiilin toiminnanohjausjärjestelmän erityisvaatimuksiin saatiin tuloksena aikaan soveltamisohjeet, joiden mukaan mobiilin toiminnanohjausjärjestelmän tuotekehityksessä tärkeimpiä tekniikoita ovat Model-View-ViewModel (MVVM) -kehitysmalli yhdessä Portable Class Libraries (PCL) -luokkakirjaston kanssa, Test-driven development (TDD) -yksikkötestaus, erikokoisten näyttötilojen huomiointi, vain päivittyneiden tietojen lataus tietokannasta sekä offline-tila. Soveltamisohjeita käytettiin WiseMaster Huolto- ja WiseMaster Myynti -tuotteiden kehityksessä, ja niiden todettiin olevan varsin päteviä. Ainoastaan TDD-mallia ei hyödynnetty kohdeprojekteissa, mutta sekin olisi ollut hyödyllinen apuväline. Soveltamisohjeet kannattaa muistaa myös tulevissa projekteissa.

Kaikki tutkitut Windows-tekniikat eivät sovellu käytettäväksi erityyppisiin projekteihin. Voidaan kuitenkin todeta, että projektin monimutkaistuessa tekniikoiden hyödyllisyys korostuu hyvin nopeasti. MVVM on hyvä esimerkki tällaisesta tekniikasta: Pienessä projektissa se on turhan raskas, mutta vähänkin suuremmassa projektissa se voi osoittautua merkittäväksi resurssien säästöksi. Windows-tekniikoita ei kuitenkaan tule laiminlyödä pienissäkään projekteissa. Pienet asiat, kuten ohjelman sujuva käyttö erikokoisissa näyttötiloissa, voi osoittautua erittäin arvokkaaksi käyttäjäkokemuksen kannalta.

LÄHTEET

1. Burela, David 2011. BUILD keynote day 1 - Metro experience with Jupiter, XAML and HTML5/JS. Saatavissa: http://davidburela.wordpress.com/2011/09/14/build-keynote-day-1metro-experience-with-jupiter-xaml-and-html5js/. Hakupäivä 1.2.2013.

2. Ferrari, Valentina - Seong, Christina 2012. Five Things to Know When Designing a Windows 8 App. Saatavissa: http://uxmag.com/articles/five-things-to-know-when-designing-a-windows-8-app. Hakupäivä 1.2.2013.

3. Warren, Tom 2012. Microsoft now using 'Modern UI Style' to refer to Windows 8 'Metro Style' apps. Saatavissa: http://www.theverge.com/2012/8/10/3232921/microsoft-modern-ui-style-metro-style-replacement. Hakupäivä 1.2.2013.

4. Microsoft 2013. Dev Center – Windows Store apps. Saatavissa: http://msdn.microsoft.com/en-US/windows/apps. Hakupäivä 2.4.2013.

5. Likness, Jeremy 2010. Model-View-ViewModel (MVVM) Explained. Saatavissa: http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained. Hakupäivä 28.1.2013.

6. Smith, Josh 2009. WPF Apps With The Model-View-ViewModel Design Pattern. Saatavissa: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx. Hakupäivä 30.1.2013.

7. Joel 2011. MVVM vs MVP vs MVC: The differences explained. Saatavissa: http://joel.inpointform.net/software-development/mvvm-vs-mvp-vs-mvc-the-differences-explained/. Hakupäivä 31.1.2013.

8. Likness, Jeremy 2009. Simplifying Asynchronous Calls in Silverlight using Action. Saatavissa: http://csharperimage.jeremylikness.com/2009/12/simplifying-asynchronous-calls-in.html. Hakupäivä 28.1.2013.

9. Likness, Jeremy 2010. Introducing the Visual State Aggrogator. Saatavissa: http://csharperimage.jeremylikness.com/2010/03/introducing-visual-state-aggregator.html. Hakupäivä 28.1.2013.

10. Likness, Jeremy 2010. Animations and View Models: IAnimationDelegate. Saatavissa: http://csharperimage.jeremylikness.com/2010/03/animations-and-view-models.html. Hakupäivä 28.1.2013.

11. Rishi 2010. Reverse ICommands for MVVM. Saatavissa: http://www.orktane.com/blog/post/2010/01/07/reverse-icommands-for-mvvm.aspx. Hakupäivä 28.1.2013.

12. Van Beek, Alex 2010. Silverlight 4: using the VisualStateManager for state animations with MVVM. Saatavissa: http://blogs.infosupport.com/silverlight-4-using-the-visualstatemanager-for-state-animations-with-mvvm/. Hakupäivä 28.1.2013.

13. Crump, Michael 2013. Using Portable Class Libraries with Windows 8 / Windows Phone 8 MVVM. Saatavissa: http://mobile.dzone.com/articles/using-portable-class-libraries. Hakupäivä 31.1.2013.

14. Plaisted, Daniel 2012. How to Make Portable Class Libraries Work for You. Saatavissa: http://blogs.msdn.com/b/dsplaisted/archive/2012/08/27/how-to-make-portable-class-libraries-work-for-you.aspx. Hakupäivä 31.1.2013.

15. Sabins, Mahesh 2012. Using Portable Class Library in .NET 4.5 and Visual Studio 2012. Saatavissa: http://www.dotnetcurry.com/ShowArticle.aspx?ID=843. Hakupäivä 31.1.2013.

16. Microsoft 2013. Managed Extensibility Framework (MEF). Saatavissa: http://msdn.microsoft.com/en-us/library/dd460648.aspx. Hakupäivä 1.2.2013.

17. Foley, Mary Jo 2012. Windows 8's delivery date: October 26. Saatavissa: http://www.zdnet.com/windows-8s-delivery-date-october-26-7000001158/. Hakupäivä 21.1.2013.

18. Angiulo, Michael - Larson-Green, Julie - Leblond, Antoine - Reller, Tami - Sinofsky, Steven 2012. Windows 8 Consumer Preview. Saatavissa: http://www.microsoft.com/en-us/news/exec/ssinofsky/2012/02-29Windows8.aspx. Hakupäivä 21.1.2013.

19. Grabham, Dan 2012. Windows 8 vs Windows 7: 8 ways it's different. Saatavissa: http://www.techradar.com/news/software/operating-systems/windows-8-vs-windows-7-8-ways-its-different-1025285. Hakupäivä 21.1.2013.

20. Dell Tech Zone staff 2012. Windows 8 Live Tiles explained. Saatavissa: http://www.pcadvisor.co.uk/opinion/windows/3417778/windows-8-live-tiles-explained/. Hakupäivä 21.1.2013.

21. Yegulalp, Serdar 2012. 9 Windows Start menus for Windows 8. Saatavissa: http://www.infoworld.com/d/microsoft-windows/9-windows-start-menus-windows-8-208963. Hakupäivä 21.1.2013.

22. Branscombe, Mary 2012. Windows 8 review: Use with a touchscreen. Saatavissa: http://www.techradar.com/reviews/pc-mac/software/operating-systems/windows-8-1093002/review/7. Hakupäivä 21.1.2013.

23. Sinun aloitusnäyttösi. 2013. Saatavissa: http://windows.microsoft.com/fi-FI/windows-8/new-look. Hakupäivä 21.1.2013.

24. Branscombe, Mary 2012. Windows 8 review: Performance. Saatavissa: http://www.techradar.com/reviews/pc-mac/software/operating-systems/windows-8-1093002/review/9#articleContent. Hakupäivä 21.1.2013.

25. Segan, Sascha 2010. Hands On with Windows Phone 7 Series. Saatavissa: http://www.pcmag.com/article2/0,2817,2359245,00.asp. Hakupäivä 21.1.2013.

26. Ricker, Thomas 2010. Microsoft announces ten Windows Phone 7 handsets for 30 countries: October 21 in Europe and Asia, 8 November in US. Saatavissa: http://www.engadget.com/2010/10/11/microsoft-announces-ten-windows-phone-7-handsets-for-30-countrie/. Hakupäivä 21.1.2013.

27. Ziegler, Chris 2010. Windows Phone 7: the complete guide. Saatavissa: http://www.engadget.com/2010/03/18/windows-phone-7-series-the-complete-guide/. Hakupäivä 21.1.2013.

28. Microsoft 2012. Technical certification requirements for Windows Phone. Saatavissa: http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh184840(v=vs.105).aspx. Hakupäivä 21.1.2013.

29. Vijay 2010. Windows Phone 7 - Application Submission and Certification Process. Saatavissa: http://www.msigeek.com/5654/windows-phone-7-application-submission-and-certification-process. Hakupäivä 21.1.2013.

30. Bancila, Marius 2011. Windows Phone 7 Quick Tutorials: Part 5 - Pivot and Panorama. Saatavissa: http://www.codeguru.com/csharp/.net/net_silverlight/article.php/c18413/Windows-Phone-7-Quick-Tutorials-Part-5--Pivot-and-Panorama.htm. Hakupäivä 21.1.2013.

31. Laberge, Paul 2011. Like a Boss! Understanding the Difference between Panorama and Pivot in Windows Phone. Saatavissa: http://blogs.msdn.com/b/cdnmobiledevs/archive/2011/12/06/like-a-boss-understanding-the-difference-between-panorama-and-pivot-in-windows-phone.aspx. Hakupäivä 24.1.2013.

32. Miller, Paul 2010. Microsoft confirms Windows Phone 7 manufacturers: ASUS, Dell, HTC, LG and Samsung all on board. Saatavissa: http://www.engadget.com/2010/07/22/microsoft-confirms-windows-phone-7-manufacturers-asus-dell-ht/. Hakupäivä 21.1.2013.

33. Kurri, Sampsa 2012. Nokia MeeGon tarina. Saatavissa: http://taskumuro.com/artikkelit/nokia-meegon-tarina,7. Hakupäivä 21.1.2013.

34. Thuret, Romain - Alzieu, Vincent 2011. Nokia Presents Lumia 800 and Lumia 710: First 'Real' Windows Phones. Saatavissa: http://www.digitalversus.com/nokia-presents-lumia-800-lumia-710-real-windows-phones-n21741.html. Hakupäivä 21.1.2013.

35. Nokia 2013. Nokia Lumia 900. Saatavissa: http://www.nokia.com/fi-fi/tuotteet/puhelimet/lumia900/. Hakupäivä 21.1.2013.

36. Cooper, Daniel 2013. Samsung ATIV Tab review: the Windows RT tablet you'll never find in the US. Saatavissa: http://www.engadget.com/2013/02/04/samsung-ativ-tab-review/. Hakupäivä 7.2.2013.

37. Microsoft 2013. Windows RT: usein kysyttyjä kysymyksiä. Saatavissa: http://windows.microsoft.com/fi-FI/windows/windows-rt-faq. Hakupäivä 7.2.2013.

38. Stables, James 2013. Microsoft Surface RT review. Saatavissa: http://www.techradar.com/reviews/pc-mac/tablets/microsoft-surface-rt-1085839/review. Hakupäivä 7.2.2013.

39. Microsoft 2013. Surface. Saatavissa: http://www.microsoft.com/Surface/en-US. Hakupäivä 7.2.2013.

40. Agrawal, Vishwani D. - Kime, Charles R. - Saluja, Kewal K. 1993. A Tutorial on Built-In Self-Test. Saatavissa: http://users.soe.ucsc.edu/~larrabee/ce224/tutorialBIST1.pdf. Hakupäivä 6.2.2013.

41. Engelin, Thomas 2006. Testable Software Design. Saatavissa: http://www.itancan.com/itancan-docs/TestableSoftwareDesign.pdf. Hakupäivä 6.2.2013.

42. Palermo, Jeffrey 2006. Guidelines for Test-Driven Development. Saatavissa: http://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx. Hakupäivä 6.2.2013.

43. Elinext Group 2013. Test Driven Development: Best Practises and Benefits of Using Unit Testing on Real Life Projects. Saatavissa: http://www.elinext.com/test-driven-development. Hakupäivä 8.2.2013.

44. Hannula, Joonas - Pitkänen, Pekka - Rahja, Esa 2013. Projektipalaveri 4.2.2013.

45. Eklund, Petri - Hannula, Joonas - Pitkänen, Pekka 2013. Projektipalaveri 7.2.2013.

46. Hannula, Joonas - Pitkänen, Pekka - Rahja, Esa 2013. Projektipalaveri 8.3.2013.

47. Hannula, Ari - Hannula, Joonas - Pitkänen, Pekka - Rahja, Esa - Wiik, Tommi 2013. Projektipalaveri 22.2.2013.